Contents

1. What Does it Mean to Learn?
2. Some Canonical Learning Problems
3. The Decision Tree Model of Learning
4. Formalizing the Learning Problem
5. Inductive Bias: What We Know Before the Data Arrives
6. Not Everything is Learnable
7. Underfitting and Overfitting
8. Separation of Training and Test Data
9. Models, Parameters and Hyperparameters

Decision Trees

Machine Learning
Youngjoong Ko
Dept. of Computer Engineering, Dong-A University

What does it mean to learn in human?

History

Alice

Generalization

Exam

Memorization or Experience

Machine Learning

What does it mean to learn?

Test Data

Learning Algorithm

Predict

Result

Training Data

Feature Instance
**Canonical Learning Problems**

- **Regression**: trying to predict a real value
  - predict the value of a stock tomorrow given its past performance
- **Binary Classification**: trying to predict a simple yes/no response
  - predict whether Alice will enjoy a course or not
- **Multiclass Classification**: trying to put an example into one of a number of classes
  - predict whether a news story is about entertainment, sports, politics, religion
- **Ranking**: trying to put a set of objects in order of relevance
  - predicting what order to put web pages in, in response to a user query

**Principle**

- If training data were given, then we would make the Decision Tree by learning from Training data

**Constraints**

- How get "question" from each node?
- How well would I have done?
- How leaf node decide class?

**Examples of Decision Tree**

```
Tree

Sunny  Overcast  Rain

Humidity
High   Normal  Weak
Yes    No      Yes

Outlook

Strong  Normal  Weak

Wind

Yes    No
```

**Generate Tree**

```
Training Data
Generate Tree
Test Data
```

**Progress**
What’s mean impurity?

**Question of node**

- **T** is current node
- **X** is set of features
- **a** is value in feature **x_i**

Then candidates of question are

Example: 혈액형이 b?

- **Entropy impurity**: \( m(T) = -\sum_{i} P(a_i | T) \log_2 P(a_i | T) \)
- **Gini impurity**: \( m(T) = \sum_{i} P(a_i | T)^2 = \sum_{i} P(a_i | T)P(\neg a_i | T) \)
- **Misclassification impurity**: \( m(T) = 1 - \max P(a_i | T) \)

**Example**

<table>
<thead>
<tr>
<th>Label</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>x_1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>x_2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x_3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Compute Impurity**

- **Entropy impurity**: \( m(T) = -\frac{3}{9} \log_2 \left( \frac{3}{9} \right) = 0.5277 \)
- **Gini impurity**: \( m(T) = 1 - \sum_{i} P(a_i | T)^2 = 0.5333 \)
- **Misclassification impurity**: \( m(T) = 1 - \frac{4}{9} = 0.556 \)
Example

1. Generate candidate question
   - Example (8): [1:7]의 점수 (1 = 미지, 2 = 스코프, 3 = 교수, 4 = 의사, 5 = 공무원, 6 = NGO, 7 = 부서)
   - 선호 점수 (8): [1:5]의 점수 (1 = 의료, 2 = 전자 제품, 3 = 스코프 응용, 4 = 일반)
   - 평균 (9): 싱글

2. Questions is:
   - $x_1$에 의한 주요 질문: $x_1 = 1? \quad x_1 = 2? \quad x_1 = 3? \quad x_1 = 4? \quad x_1 = 5? \quad x_1 = 6? \quad x_1 = 7$
   - $x_2$에 의한 주요 질문: $x_2 = 1? \quad x_2 = 2? \quad x_2 = 3? \quad x_2 = 4? \quad x_2 = 5$
   - $x_3$에 의한 주요 질문: $x_3 = 1? \quad x_3 = 2? \quad x_3 = 3? \quad x_3 = 4? \quad x_3 = 5$
   - $x_4$에 의한 주요 질문: $x_4 = 6.7? \quad x_4 = 9.27? \quad x_4 = 57.55? \quad x_4 = 86.55? \quad x_4 = 70.3? \quad x_4 = 80.75? \quad x_4 = 90.5? \quad x_4 = 97.25$

Decision Tree Train

- Generate questions from Node T, Feature $X_j$
- Check Impurity of all candidates
- Select Max Impurity
- Decrease Value of q

1. $\text{Impurity} \leq 0$
2. Cannot split $X_j$ [sample lower than threshold]
3. $q$ is lower than threshold

Is $T$ satisfied?

1. Split $X$ into $X_{\text{left}}$ and $X_{\text{right}}$ by $q$
2. Generate $T_{\text{left}}$ and $T_{\text{right}}$
3. $\text{DecisionTreeTrain}(T_{\text{left}}, X_{\text{left}})$
4. $\text{DecisionTreeTrain}(T_{\text{right}}, X_{\text{right}})$

Impurity decrement

$\Delta \text{Impurity}(T) = \text{Impurity}(T) - \frac{|X_{\text{left}}|}{|X|} \cdot \text{Impurity}(T_{\text{left}}) - \frac{|X_{\text{right}}|}{|X|} \cdot \text{Impurity}(T_{\text{right}})$

Example

Decision Tree Train

Algorithm 1 $\text{DecisionTreeTrain}(data, \text{remaining features})$
1. $\text{guess} = \text{most frequent answer in data}$ // default answer for this data
2. if labels in data are unanimous then
   return $\text{Leaf}($guess$)$ // base case: no need to split further
3. else if remaining features is empty then
   return $\text{Leaf}($guess$)$ // base case: cannot split further
4. else
   for all $f \in \text{remaining features}$ do
5. if $f$ is the feature with maximal $\text{score}(f)$
6. YES $\leftarrow \text{the subset of data on which } f = \text{yes}$
7. NO $\leftarrow \text{the subset of data on which } f = \text{no}$
8. if $\text{score}(f) \neq 0$
9. return $\text{Leaf}($YES remaining features $\setminus \{f\})$
10. return $\text{Leaf}($NO remaining features $\setminus \{f\})$
11. return $\text{DecisionTreeTrain}($YES, remaining features $\setminus \{f\}$)$
12. return $\text{DecisionTreeTrain}($NO, remaining features $\setminus \{f\}$)$
13. end if
14. end for
**Decision Tree Test**

DecisionTreeTest(Tree R, TestData x)

\[ T = \text{Node of } R \]

\[ \text{Is } T \text{ Leaf ?} \]

\[ \text{NO} \]

\[ X \text{ is } w(\text{case } T) \]

\[ \text{YES} \]

\[ r \text{ (yes or no) } = \text{predict } T\text{'s question from } x \]

\[ \text{Is } r \text{ yes?} \]

\[ \text{NO} \]

\[ \text{DecisionTreeTest} \left( T_{\text{left}}, x \right) \]

\[ \text{YES} \]

\[ \text{DecisionTreeTest} \left( T_{\text{right}}, x \right) \]

---

**Formalizing the Learning Problem**

- There are several issues when formalizing the notion of learning:
  - The performance of the learning algorithm should be measured on unseen "test" data.
  - The way in which we measure performance should depend on the problem we are trying to solve.
  - There should be a strong relationship between the data that our algorithm sees at training time and the data it sees at test time.
- Loss function:
  - To tell us how “bad” a system’s prediction is in comparison to the truth. In particular
  - If \( y \) is the truth and \( \hat{y} \) is the system’s prediction, then \( f(y, \hat{y}) \) is a measure of error.

---

**Algorithm 2** DecisionTreeTest(tree, test point)

1. If tree is of the form Leaf(guess) then
   - return guess
2. else if tree is of the form Node(left, right) then
   - if \( x \) is yes in test point then
     - return DecisionTreeTest(left, test point)
   - else
     - return DecisionTreeTest(right, test point)
   - end if
- end if

---

**Formalizing the Learning Problem**

- For three of the canonical tasks discussed above, we might use the following loss functions:
  - **Regression**: squared loss \( f(y, \hat{y}) = (y - \hat{y})^2 \)
    or absolute loss \( f(y, \hat{y}) = |y - \hat{y}| \).
  - **Binary Classification**: zero/one loss \( f(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \).
  - **Multiclass Classification**: also zero/one loss.
    - Note that the loss function is something that you must decide on based on the goals of learning.
- Now that we have defined our loss function, we need to consider where the data comes from.
Formalizing the Learning Problem

- There is a probability distribution $D$ over input/output pairs
  - This is often called the data generating distribution
  - If we write $x$ for the input and $y$ for the output, then $D$ is a distribution over $(x, y)$ pairs

- Formally, it's expected loss $\epsilon$ over $D$ with respect to $f$ should be as small as possible
  - we don't know what $D$ is!

$$\epsilon \triangleq \mathbb{E}_{(x,y) \sim D} [f(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x,y) \ell(y, f(x))$$

Suppose that we denote our training data set by $D$

- The training data consists of $N$-many input/output pairs, $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

- Given a learned function $f$, we can compute our training error

$$\hat{\epsilon} \triangleq \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n))$$

  - our training error is simply our average error over the training data

Given a loss function $\ell$ and a sample $D$ from some unknown distribution $D$, you must compute a function $f$ that has low expected error $\epsilon$ over $D$ with respect to $\ell$

Inductive Bias

- What we know before the data arrives
- Preference type A
- Preference type B
- Preference for one distinction over another is a bias that different human learners have
- inductive bias: in the absence of data that narrow down the relevant concept

We will not allow the trees to grow beyond some pre-defined maximum depth, $d$

- That is, once we have queried on $d$-many features, we cannot query on any more and must just make the best guess we can at that point

- The key question is: What is the inductive bias of shallow decision trees?
- Roughly, their bias is that decisions can be made by only looking at a small number of features
Not Everything is Learnable

- There are many reasons why a machine learning algorithm might fail on some learning task.
- There could be noise in the training data.
  - Noise can occur both at the feature level and at the label level.
- Some example may not have a single correct answer.
- In the inductive bias case, it is the particular learning algorithm that you are using that cannot cope with the data.

Overfitting and Underfitting

- Overfitting is when you pay too much attention to idiosyncrasies of the training data, and aren’t able to generalize well.
- Underfitting is when you had the opportunity to learn something but didn’t.
  - This is also what the empty tree does.

Separation of Training and Test Data

- The easiest approach is to set aside some of your available data as “test data” and use this to evaluate the performance of your learning algorithm.
- If you have collected 1000 examples, You will select 800 of these as training data and set aside the final 200 as test data.
- Occasionally people use a 90/10 split instead, especially if they have a lot of data.
- They cardinal rule of machine learning is: **Never ever touch your test data!**

Models, Parameters and Hyperparameters

- The general approach to machine learning, which captures many existing learning algorithms, is the modeling approach.
- For most models, there will be associated parameters.
- **Hyperparameter**: we can adjust between underfitting and overfitting by the DecisionTreeTrain function so that it stop recursing.
  - choosing hyperparameters: choose them so that they minimize training error.
- The job of the development data is to allow us to tune hyperparameters.
Some people call this “validation data” or “held-out data.”

Split your data into 70% training data, 10% development data and 20% test data.

For each possible setting of your hyperparameters
  - Train a model using that setting of hyperparameters on the training data
  - Compute this model’s error rate on the development data

From the above collection of models, choose the one that achieved the lowest error rate on development data.

Evaluate that model on the test data to estimate future test performance.